— Lecture Notes —

First Look at Chart Parsing

1 A *chart* (or *active chart*) represents a generalization of the well-formed substring table of the CYK algorithm. The essential generalization is that we now maintain entries for substring analyses that are only partial. In this way we represent our hypotheses and predictions as well as completed categories.

In a way this is like GLC parsing, in the sense that we are dealing once again in predictions and completions. There however we were working at the level of individual categories. Here the essential unit will be the grammar rule.

2 Items The main tool we will be using is generally referred to as an *item*. The core of a parsing item is a *dotted rule*. A dotted rule is an ordinary grammar rule in which a dot symbol (here $\bullet$) appears somewhere on the right hand side. The position of the dot is intended to indicate how much of the left hand side constituent has been found in the input. For example, the dotted rule

$$NP \longrightarrow \bullet \; Det \; N$$

should indicate a pure hypothesis: we think we will find an *NP*, but we have yet to find any actual evidence for one. The dotted rule

$$NP \longrightarrow Det \bullet N$$

on the other hand should indicate that we have found the *Det*, but not yet the *N*. Finally, the dotted rule

$$NP \longrightarrow Det \; N \bullet$$

should indicate that we have found a complete *NP*.

An item is then a triple, associating a dotted rule with two sentence positions: the position to the immediate left of the first subconstituent and the position of the dot. In the special case of dotted rules of the form $A \longrightarrow \bullet \, \alpha$, the two positions will be equal. In the case of a dotted rule of the form $A \longrightarrow \alpha \bullet$, the item is essentially like the contents of a cell in a well-formed substring table. Such items are called *completed items*. All other items are called *active items*. Because the chart has a graph like structure, items are also called *arcs* or *edges* in the chart.

3 Fundamental Rule From the above informal discussion, it should be clear that a sentence has been recognized by a chart recognizer just in case we have managed to construct an item of the form $\langle S \longrightarrow \alpha \bullet, 0, |w| \rangle$, for input string w. The initial state of the recognizer will be represented by items of the form $\langle S \longrightarrow \bullet \, \alpha, 0, 0 \rangle$.

The fundamental problem in parsing with an active chart is thus going to be the problem of *moving the dot.* The rule for doing this—generally called the *fundamental rule*—is as follows.

$$\frac{\langle A \longrightarrow \alpha \bullet B\beta, i, j\rangle \quad \langle B \longrightarrow \gamma \bullet, j, k\rangle}{\langle A \longrightarrow \alpha B \bullet \beta, i, k\rangle}$$

Essentially this just says: if we're looking for a B, and we find one, then we get to record this fact in the chart by moving the dot across the B.

In the special case that $\beta = \epsilon$, we construct a new completed edge, and that in its own turn can be used as an assumption in new applications of the fundamental rule.

It is important to remember that "moving the dot" actually involves the construction of a new item, and not the modification of the old item. The old active item remains in the chart since, after all, we might discover a new way to construct a B, from j to k', and we would want to be able to make use of that edge as well.

4 Initialization Each word in the input sentence allows us to construct a (rather small) completed edge in the chart. So one step in initializing the chart is essentially like the first step in CYK parsing.

In addition to these completed edges we need some active edges in order to get the fundamental rule going.

In fact, we have considerable freedom in how we introduce active edges, and variations in how we do this will lead to a variety of different chart parsing strategies. Actually, well-formed substring tables can also be used in a variety of parsing strategies. The CYK algorithm represents bottom up tabular parsing, but top down tabular parsing exists as well, and so on. But the chart is a more useful auxiliary data structure than the well-formed substring table, so we hurried past that fact in order to get where we are now.

5 Bottom Up Chart Recognition Actually, it would be more accurate to call this left corner recognition. The idea is to generate active arcs by considering what completed arcs we have already, and finding rules of which they are the left corner. So, given an item like $\langle B \longrightarrow \beta \bullet, i, j\rangle$ and a grammar rule like $A \longrightarrow B\alpha$, we construct a new item $\langle A \longrightarrow B \bullet \alpha, i, j\rangle$.

(Other treatments of this subject introduce items of the form $\langle A \longrightarrow \bullet B\alpha, i, i\rangle$, to which the fundamental rule will apply immediately to yield the item I have given above.)

6 Top Down Chart Recognition We have already noted that at the start of recognition the item $\langle S \longrightarrow \bullet \alpha, 0, 0\rangle$ expresses a true fact about the recognition procedure: at the start, we expect an S but we haven't got any of it yet.

If $\alpha = A\beta$, then one prediction we can make is that we may find our S by first finding an A. Given a rule $A \longrightarrow \gamma$, we add to the chart the item

$\langle A \longrightarrow \bullet\, \gamma, 0, 0\rangle$. Then we iterate this procedure for all the ways of expanding the left corner of γ.

Note that a rule of the form $B \longrightarrow v$, where v is a string of terminals, yields no predictions, since nothing in v occurs on the left hand side of any rule, so this procedure does eventually terminate.

7 Left Recursion Keep in mind that the chart is a generalization of the well-formed substring table. In particular, it is meant to save us time by preventing us from recomputing the same fact more than once.

So suppose our grammar contains a rule $A \longrightarrow A\alpha$. Then, having constructed the item $\langle A \longrightarrow \bullet\, A\alpha, 0, 0\rangle$, and adding it to the chart, our top down strategy will lead us to construct from its left corner a further initial item, using the same rule (left recursively). But this item will also look like $\langle A \longrightarrow \bullet\, A\alpha, 0, 0\rangle$. That item is already in the chart, so we do not add it again, nor do we attempt to propagate its consequences.

In this way the top down chart recognition strategy (which, by the way, is Earley's Algorithm) avoids the problems posed by left recursion.

By the reuse of active edges, an edge of the form $\langle NP \longrightarrow \bullet\, Det\ N, 0, 0\rangle$ can be used twice in a sentence beginning *the villain's henchman* ... (assuming a rule like $Det \longrightarrow NP$ *'s*): once as part of the analysis of *the villain* and once as part of the analysis of *the villain's henchman*. So the fact that we only add the edge to the chart once does not limit our ability to parse recursively left branching constituents.

8 Control The control structure for a chart recognizer is once again agenda-driven.

I spoke above of adding edges to the chart, but in fact new edges get added to the agenda. Then, on each cycle through the main loop, an edge is removed from the agenda, added to the chart *if it is not there already*, and its consequences are computed.

Once again we are in the situation of needing (a variant of) the semi-naive algorithm: the agenda item is just a single edge, and the operator in question, defined by the Fundamental Rule, is binary. So as usual we have to consider using the agenda item both as the left and right argument of this operation.

9 There are some important special features that we can take advantage of here that are not true in general for the semi-naive algorithm. In particular, the right term must be a completed edge and the left term an active edge. So, given any particular agenda item, its role as a left or right argument of the "fundamental operator" is fixed. (Also we needn't ever worry about combining an edge with itself.)

In the first case, where the agenda item is an active edge, and there are completed edges for it to combine with, the operation is called *scanning*. In the second case, where the agenda item is a completed edge and there is an active edge for it to combine with (this is always the case going top down) is called *completing*.

10 There is also a derived *unary* operation, which varies depending on the parsing strategy.

Earley's algorithm is top down, and propagates edges in the following manner: Given that the edge picked off the agenda is active, and there is no completed edge for it to combine with, we use it to generate more predictions, top down. This operation is called *predicting*.

In bottom up chart parsing, we would fall back on the quasi-left corner rule illustrated above: If we are adding an inactive edge, then look for rules of which it is the left corner and propose active edges for those rules.